

Zipper logic

Marius Buliga

Institute of Mathematics of the Romanian Academy

P.O. BOX 1-764, RO 014700, Bucharest, Romania

Marius.Buliga@gmail.com

20.05.2013

Abstract

Zipper logic is a graph rewrite system, consisting in only local rewrites on a class of zipper graphs. Connections with the chemlambda artificial chemistry and with knot diagrammatics based computation are explored in the article.

1 Introduction

Zipper logic is a graph rewrite system. It consists in a class of graphs, called zipper graphs and a collection of moves (graph rewrites) acting on zipper graphs, introduced in section 2. Discover more at the [zipper logic tag](#) from the author's open notebook.

Zipper logic is a variant of the chemlambda [3] [1] artificial chemistry, as explained in section 3. In section 4 we prove that zipper logic is Turing universal, by showing that it can be used to implement the SKI combinatory logic. Because zipper logic, as chemlambda, has only local graph rewrites, we treat with special attention the processes of birth and death of zipper combinator graphs, which replace the global fan-out rewrite move which is needed in (this graphical version of) combinatory logic.

In section 5 we turn to the exploration of relations between zipper logic and knot diagrammatics.

2 Half-zippers, zippers and moves

Let's start by defining the zipper graphs. Such a graph is made by the basic ingredients described in Fig. 1 and Fig. 2.

Definition 2.1 *A zipper graph is an oriented graph which has as nodes:*

- $(-n)$ half-zippers, depicted in the first row of the Fig. 1; for any natural number $n \geq 1$, a $(-n)$ half-zipper is a node with $n + 2$ arrows, which are

ordered, for convenience by numbering them with $0, 0', 1, \dots, n$, such that the arrow numbered by 0 points to the node and the arrows numbered by $0', 1, \dots, n$ point away from the node;

- $(+n)$ half-zippers, depicted in the second row of the Fig. 1; for any natural number $n \geq 1$, a $(+n)$ half-zipper is a node with $n + 2$ arrows, which are ordered by numbering them with $0, 0', 1', \dots, n'$, such that the arrows numbered by $0, 1', \dots, n'$ point to the node and the arrow numbered by $0'$ points away from the node;
- (n) zippers, depicted in the third row of the Fig. 1; for any natural number $n \geq 1$, a (n) zipper is a node with $2n + 2$ arrows, which are separated into two disjoint sets of $n + 1$ arrows; the first set is formed by the arrows numbered by $0, 1', \dots, n'$, which point to the node and the second set is formed by the arrows numbered by $0', 1, \dots, n$, which point away from the node;

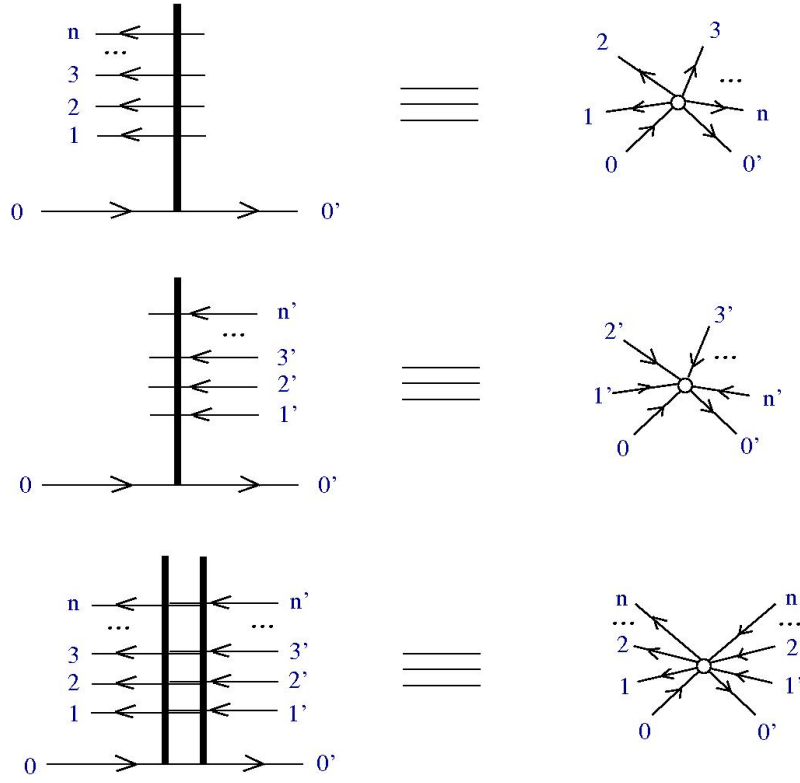


Figure 1: Zipper nodes: (1st row) $(-n)$ half-zipper, (2nd row) $(+n)$ half-zipper, (3rd row) (n) zipper.

- fanout and fanin nodes, described in Fig. 2 (a), (b), are trivalent nodes with a cyclic order of the arrows,
- termination is a univalent node, with an arrow which points to the node, described in Fig. 2 (c).

The arrows connect the nodes. We may have moreover arrows which point to one of the nodes but which have the origin not connected to any node, or we may have arrows with the origin connected to a node, but with the end free. Finally we may have arrows with both ends free, or loops, as described in Fig. 2 (c).

A zipper graph is made by a finite number of nodes, arrows and loops; it does not have to be connected.

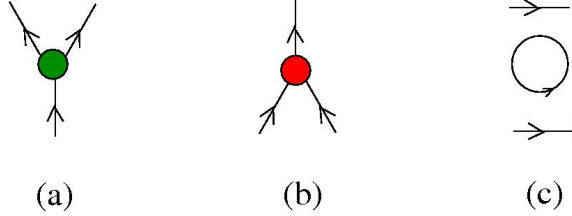


Figure 2: The other ingredients: (a) fanout node, (b) fanin node, (c) arrow, loop and termination node.

The graph rewrites of zipper logic are described in the following figures.

Definition 2.2 *The moves of zipper logic are all reversible. They act on a bounded number of nodes (but due to the fact that they act on half-zippers or zippers with an unbounded number of arrows, they may act on any number of arrows). The figures which describe the moves contain only the region, or pattern, which is subjected to the respective move. They come in several families:*

- the *CLICK* moves, Fig. 3 transform pairs of half-zippers into a zipper and possibly a half-zipper; in the Fig. 3 is described a *CLICK* move which involves a $(-n)$ half-zipper and a $(+m)$ half-zipper with $m > n$; the other cases, namely $m = n$ and $m < n$ are not shown, but they are straightforward to imagine;

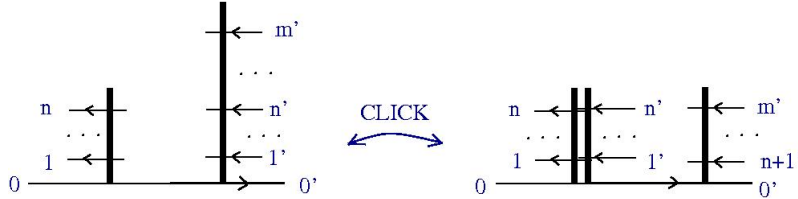


Figure 3: The *CLICK* move for $m > n$.

- the *ZIP* move, Fig. 4, is the one which gives the name to the zipper logic,



Figure 4: The *ZIP* move.

because it may be imagine as the act of unzipping a zipper, when seen from left to right, or to zip it, when seen from right to left;

- the *TOWER* moves, Fig. 5, serve to stack half-zippers,

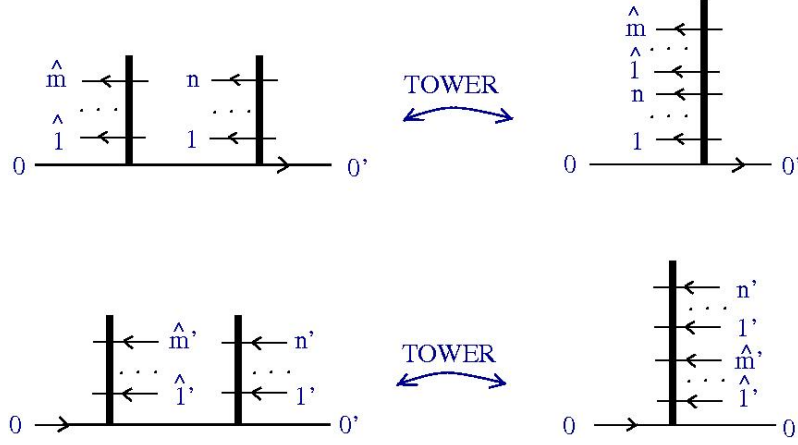


Figure 5: The *TOWER* moves.

- *CO-COMM*, *CO-ASSOC* and *FAN-IN* moves, Fig. 6, are the same as the ones from *chemlambda* [3] [1], see also section 3;

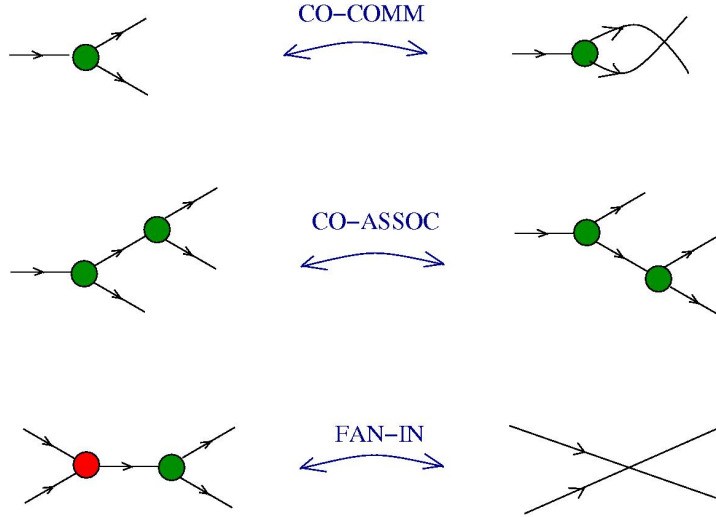


Figure 6: Some of the moves involving the other nodes: (1st row) *CO-COMM* move, (2nd row) *CO-ASSOC* move, (3rd row) *FAN-IN* move.

- the *DIST* moves are described in Fig. 7.

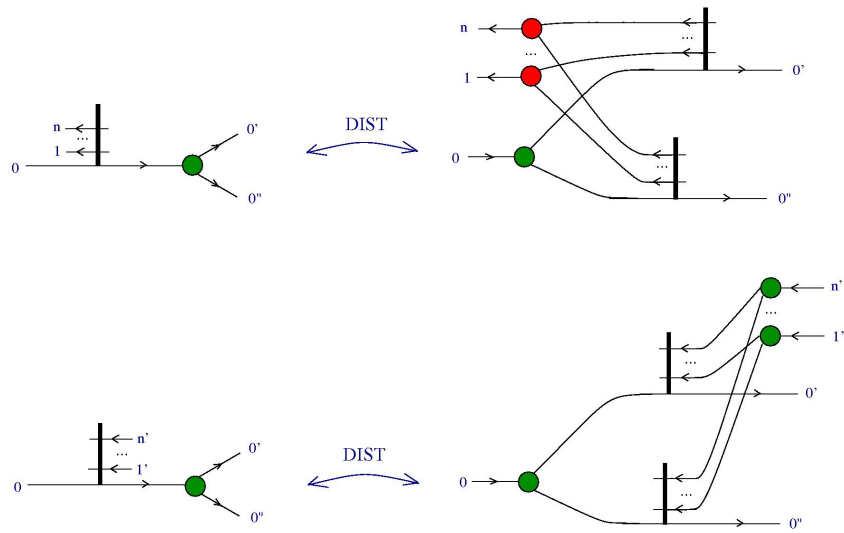


Figure 7: The DIST moves.

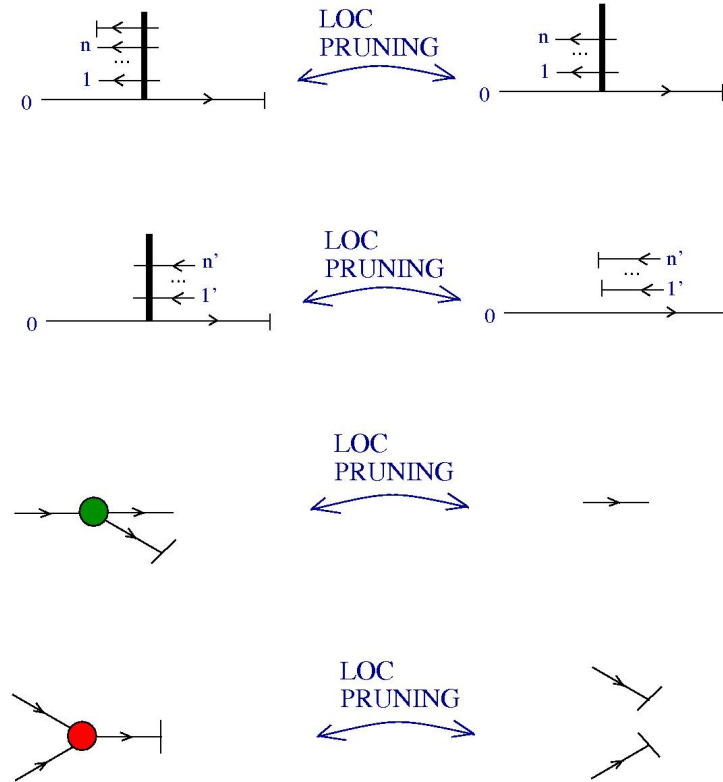


Figure 8: Local pruning moves. The first two rows describe moves for half-zippers, the last two rows describe moves for the other nodes.

- the last two rows of Fig. 8 are the *LOCAL PRUNING* moves for the fanout and fanin nodes from chemlambda, the first two rows of the mentioned figure describe the *LOCAL PRUNING* moves of half-zippers;

3 Relations to chemlambda

Chemlambda is a graph rewrite system, dressed as an artificial chemistry. It consists into a set of graphs called molecules and a collection of local moves, or graph rewrites. It has been introduced in [3] as an alternative to graphic lambda calculus, or GLC, [4]. In chemlambda there are only local moves, i.e. there is an a priori bound on the number of edges and nodes which are involved in any of the moves.

There is a distributed, decentralized computing model associated to chemlambda (or GLC), called distributed GLC [2]. A good introduction to chemlambda, which also emphasizes it's biological like self-multiplication features, is [1].

Proposition 3.1 (a) *In chemlambda, let's define half-zippers as in the Fig. 9 and let's use the CLICK move from the Fig. 3 as the definition of a zipper. Then every move from the zipper logic can be realized as a finite sequence of chemlambda moves. In particular the DIST moves for half-zippers show that they are distributors, in a generalized sense, explained in [1] section "Propagators, distributors, multipliers and guns".*

(b) *In zipper logic, let's define the lambda abstraction node as a (-1) half-zipper, the application node as a $(+1)$ half-zipper. Then the moves CLICK followed by ZIP is the beta move from chemlambda, the TOWER and CLICK moves serve to translate from zipper graphs to chemlambda molecules and the rest of the moves, used only with 1 half zippers, are exactly the moves of chemlambda.*

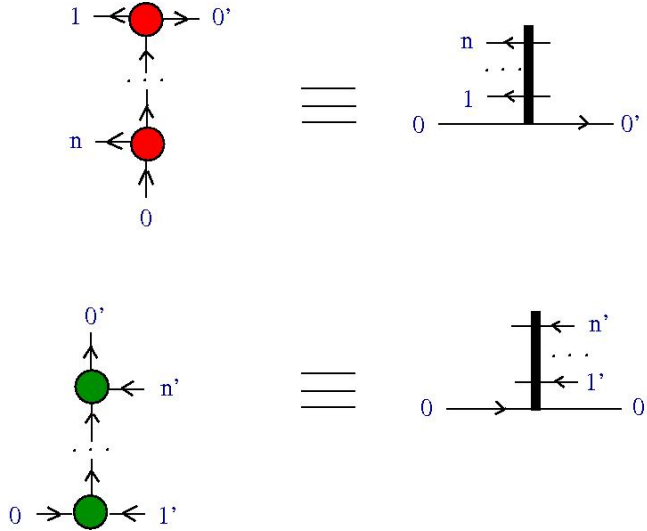


Figure 9: Zippers in chemlambda.

We leave the proof to the reader, instead of giving detailed explanations, because it is simply a matter of comparison of graph rewrites from the two formalisms. Please use [1] section "The Chemlambda formalism", Figures 1-5, as reference for the chemlambda moves.

4 Zipper combinators

From Proposition 3.1 we get that zipper logic and chemlambda are equivalent. In particular, it follows that zipper logic is Turing universal, as chemlambda, because it contains combinatory logic.

However, the zipper logic may be more intuitive than chemlambda. Let's see how the SKI system of combinators appear and function in zipper logic.

Definition 4.1 *The set of zipper combinators is the smallest set of zipper graphs with the properties:*

- *it contains the S , K , I zipper graphs defined in Fig 10,*
- *for any natural number $n > 0$ and for any $n + 1$ zipper combinators, the zipper graph obtained by connecting the out arrows of the zipper combinators to the in arrows of the $(+n)$ half-zipper is a zipper combinator.*
- *any zipper graph which is obtained by applying a zipper logic move to a zipper combinator is a zipper combinator.*

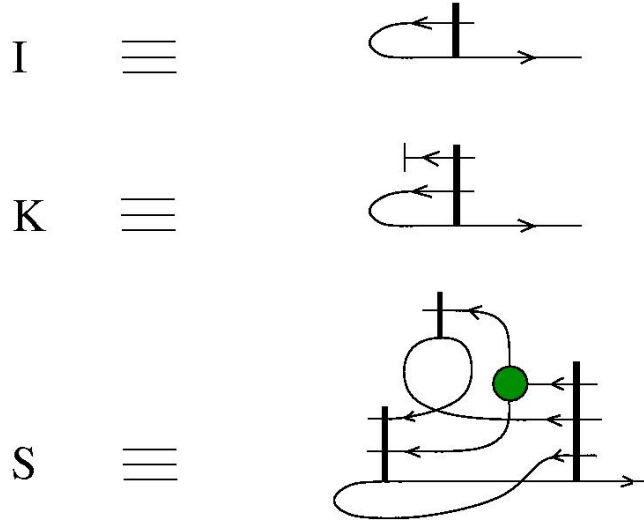


Figure 10: The S , K , I zipper combinators.

For any two zipper combinators A , B , the zipper combinator AB is obtained by connecting A and B to a $(+1)$ half-zipper. More generally, from any $n + 1$ zipper combinators A_0, \dots, A_n , we obtain the zipper combinator $(\dots(A_0 A_1) \dots A_n)$ described in Fig. 11.

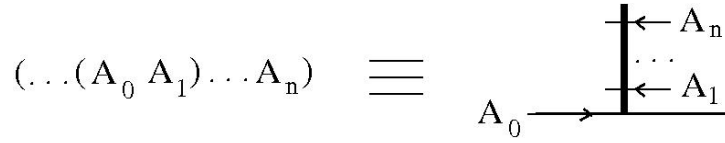


Figure 11: (+) half-zippers as concatenations of applications at right. This is compatible with Fig. 9.

Birth and death of zipper combinators. We shall need the following lemmata, which describe how zipper combinators multiply or die.

Lemma 4.2 *Any zipper combinator is a multiplier, i.e. for any zipper combinator A , the graph obtained by connecting the out arrow of A to the entry arrow of a fanout node transforms by a finite sequence of zipper logic moves into two copies of A .*

Proof. Suppose that the zipper combinator A is formed by $n + 1$ zipper combinators A_0, \dots, A_n connected to a $(+n)$ half-zipper. Use then the DIST move for (+) half-zippers from Fig. 7, and remark that if A_0, \dots, A_n are multipliers then A is a multiplier. From the definition 4.1 it follows that in order to prove our lemma, it is sufficient to prov that S , K and I zipper combinators are multipliers. But this has been shown several times in previous articles. This is related to [1] section "Propagators, distributors, multipliers and guns". In the Step 2 of the proof of Theorem 4.2 [3] there is such a detailed proof, only the formalisms differ slightly, i.e. in the mentioned reference it is used chemlambda, while here we use zipper logic, moreover there was used the BCKW system of combinators, while here it is used the SKI system of combinators. That is why we leave the completion of the rest of the proof to the interested reader. \square

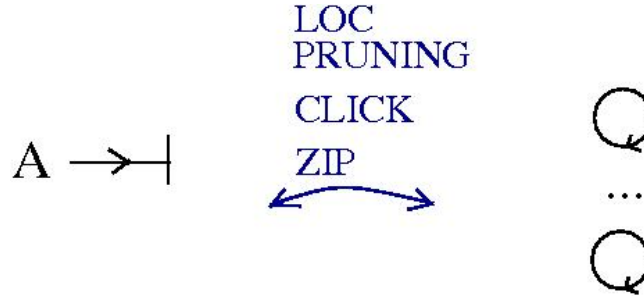


Figure 12: Every zipper combinator connected to a termination node transforms into a finite collection of loops.

Lemma 4.3 *For any zipper combinator A , the graph obtained by connecting the out arrow of A with a termination node can be reduced, by a finite number of moves, to a graph formed by a finite number of loops (or the empty graph). See Fig. 12.*

Proof. Consider the zipper combinator obtained from $n+1$ zipper combinators A_0, \dots, A_n connected to a $(+n)$ half-zipper, then connect the out arrow of this zipper combinator to a termination node. By the LOC PRUNING move from Fig. 8, second row, we can reduce this graph to the collection of A_0, \dots, A_n , each connected to a termination node.

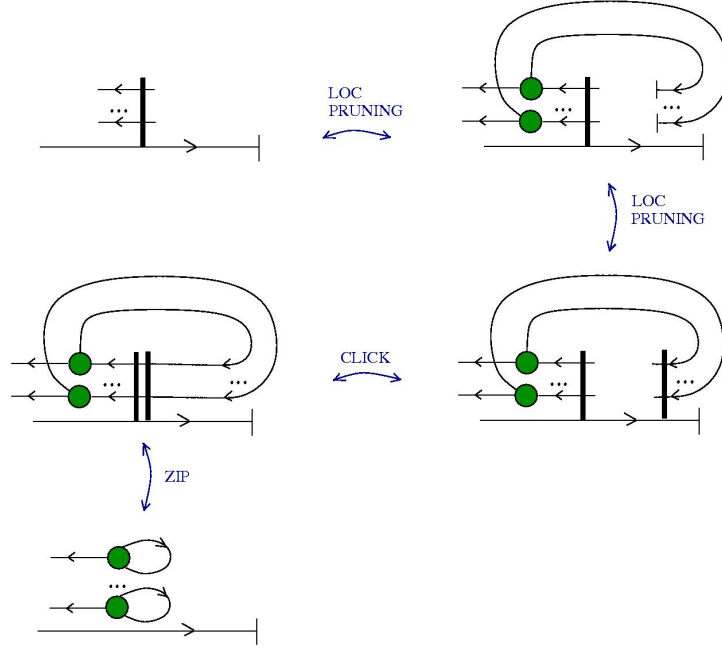


Figure 13: A succession of moves used to eliminate a $(+)$ half-zipper connected to a termination node.

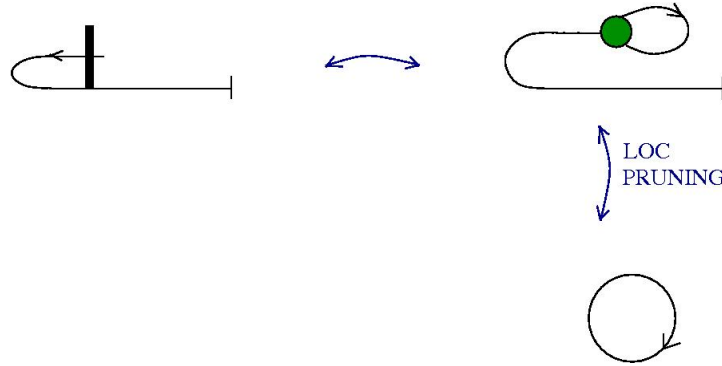


Figure 14: The I combinator connected to a termination node transforms into one loop.

For progressing further we shall use the following "trick", described in Fig. 13, which will be used to pass over a $(+)$ half-zipper connected to a termination node.

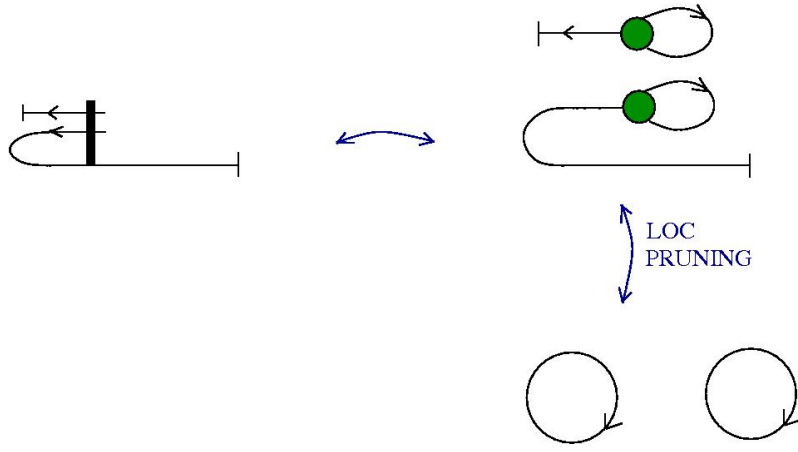


Figure 15: The K combinator connected to a termination node transforms into two loops.

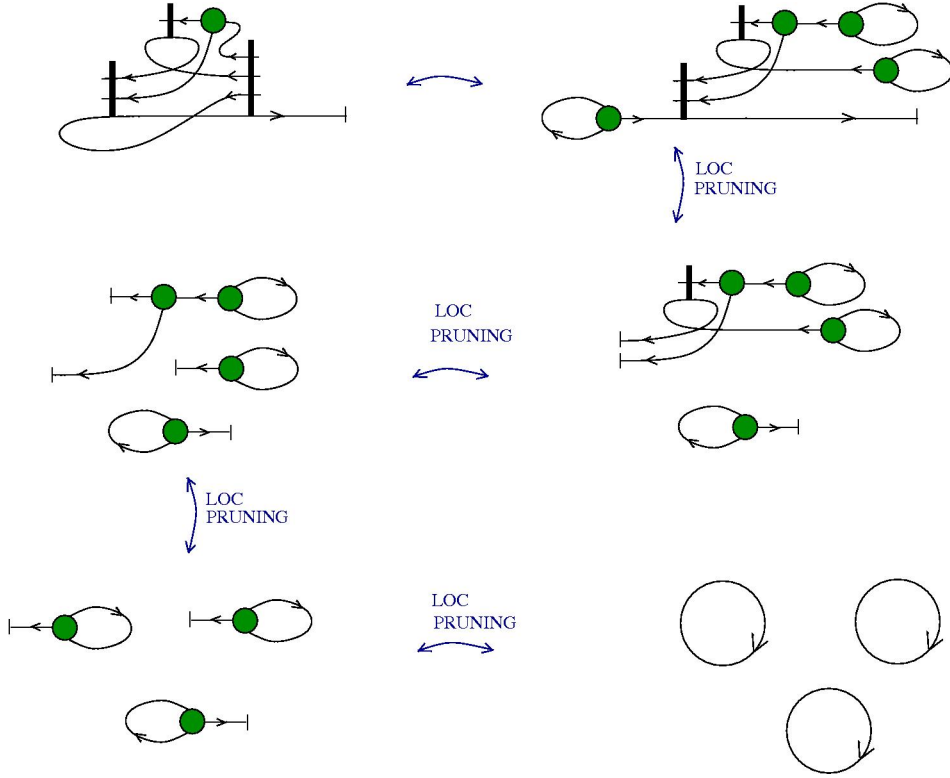


Figure 16: The S combinator connected to a termination node transforms into three loops.

In the second part of the proof we use the "trick" for the zipper combinators I , K and S connected to a termination node. In the case of the I combinator.

we can transform it into one loop, as described in Fig. 14. The first blue arrow represents the "trick".

The same trick is used for transforming the K zipper combinator, connected to a termination node, into two loops, Fig. 15.

The zipper combinator S , connected to a termination node, is transformed into three loops, starting by the same trick, Fig. 16. \square

Theorem 4.4 Consider the set of all zipper combinators, with the relation " $A = B$ " meaning that there is a finite sequence of zipper logic moves which transforms A into B and a finite number of loops, and with the operation AB as defined in the Fig. 11. Then we have the relations:

- (a) $IA = A$ for any zipper combinator A ,
- (b) $(KA)B = A$ for any zipper combinators A, B ,
- (c) $((SA)B)C = (AC)(BC)$ for any zipper combinators A, B, C ,
- (d) $(SK)K = I$.

Proof. We start by proving (a), (b) and (d). Then we prove (c) as a consequence of the fact that zipper combinators are multipliers.

The proof of (a) is given in Fig. 17.

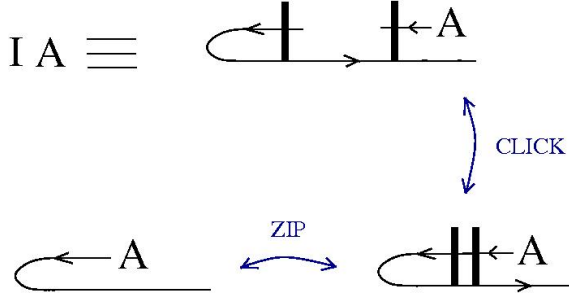


Figure 17: Proof of $IA = A$.

The proof of (b) is given in Fig. 18. On the second row of the figure, the move from right to left is the one from Lemma 4.3, which transforms the zipper combinator B connected to a termination node into a finite collection of loops.

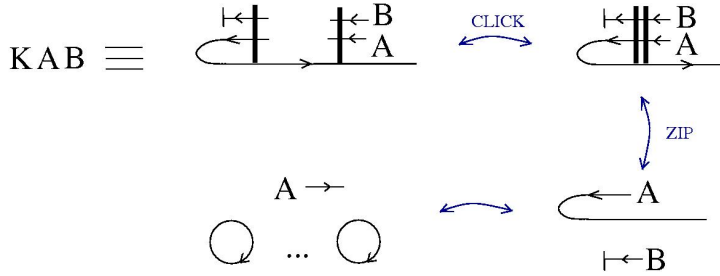


Figure 18: Proof of $(KA)B = A$.

The proof of (d) is given in Fig. 19.

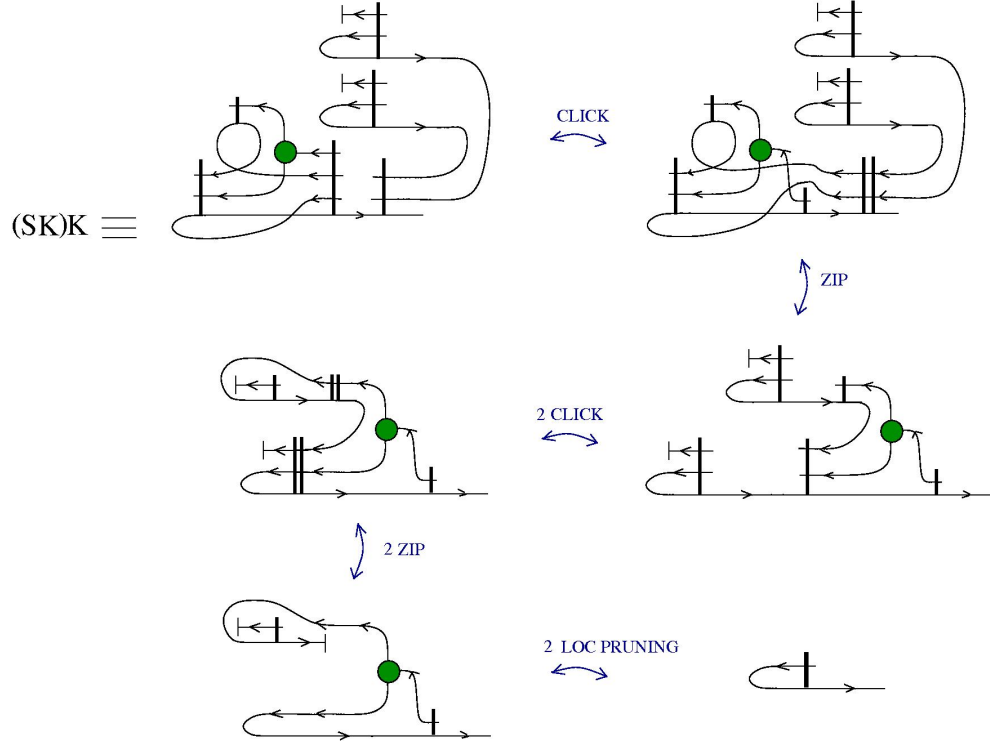


Figure 19: Proof of $(SK)K = I$.

For the proof of (c) see Fig. 20.

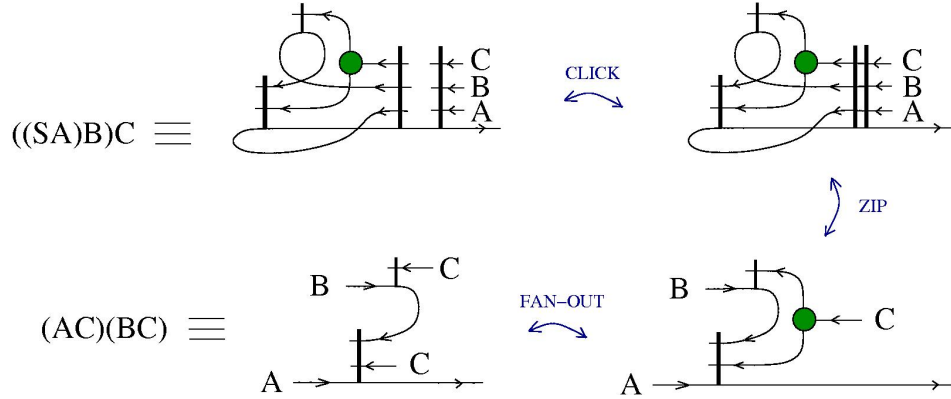


Figure 20: Proof of $((SA)B)C = (AC)(BC)$.

The "FAN-OUT" move from the second row of the figure is the one which multiplies the zipper combinator C , according to the lemma 4.2. \square

5 Zippers and knots

The fact that there are relations between knots and tangles diagrams and graph rewrites systems which are close to zipper logic has been already noticed. In the article which introduces GLC (or "graphic lambda calculus") [4], in the section 6, see also Theorem 6.1, is proved that GLC has a sector which is equivalent with the formalism of locally planar oriented crossings diagrams with the oriented Reidemeister moves. In this sector we represent an oriented crossing as a pair of two nodes, lambda abstraction and application. In [2], section 5, are discussed various connections between a topological version of GLC and knot diagrammatic notations for lambda calculus or for topological quantum computations. In the last section of [1], there is proposed another encoding of an oriented crossing, as a pair fanout and application nodes, which leads to a graph rewrite formalism over oriented knots or tangle diagrams, with a lambda abstraction trivalent node added, along with a termination univalent node.

The zipper logic formalism can be transformed into a graph rewriting formalism over knots or tangles diagrams, with the trivalent fanin and fanout nodes added, along with the univalent termination node. Indeed, let's define half-zippers like in the Fig. 21.

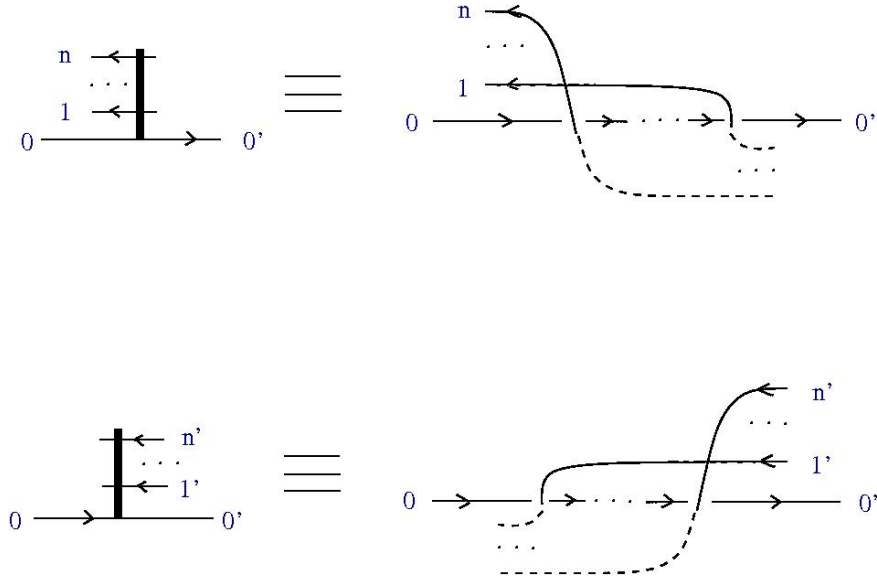


Figure 21: Definition of zippers as knot diagrams.

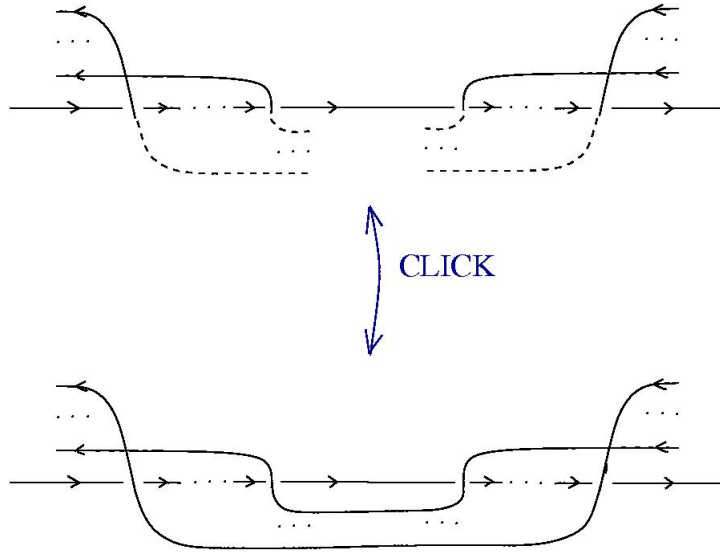


Figure 22: The CLICK move for zippers as knot diagrams.

The dotted lines represent arcs which are only virtually there. Their advantage is that we can see the CLICK move, in the realm of knot diagrams, as a move which transforms virtual arcs into real, connected arcs, see Fig. 22.

This way, the ZIP move (which is the equivalent of the graphic beta move from GLC) appears as one of the oriented Reidemeister 2 moves.

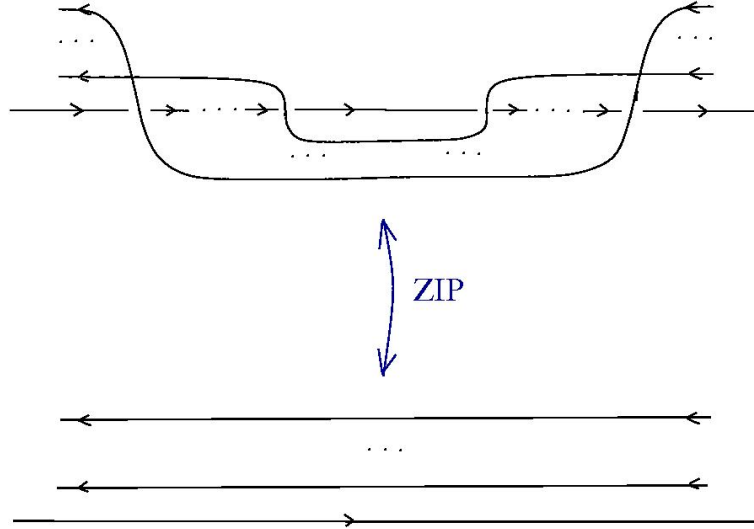


Figure 23: The ZIP move for zippers as knot diagrams is the same as R2.

Finally, the S,K,I zipper combinators appear as the following "knot combinators" from Fig. 24.

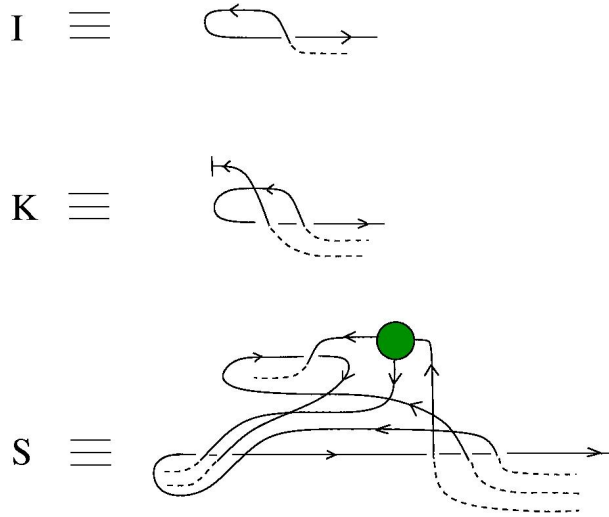


Figure 24: The S,K,I combinators as knot diagrams.

The connections between knot diagrammatics and lambda calculus started by the article Kauffman [8], where in section 5 knot diagrams are used as notations for combinatory logic terms. See also [7] where knot automata are introduced. The most recent discussion concerning diagrammatic methods for representing quantum processes and quantum computing is to be found in [10]. Another interesting, related research thread is the one of tangle machines by Carmi and Moskovich [5] [6]. In all these articles knot diagrams are used as a notational device for computations. The topology appears as related to the invariance of these computations, in the sense that knot diagrams which are related by a sequence of Reidemeister moves describe the same computation. In few words, in this case we may say that "topology does not compute, but is an invariant".

In contradistinction, in the knot diagrams sector of GLC, or in the topological version of GLC the topology does compute. This means that (some of the) reduction moves appear as graph rewrites on knot diagrams which change their topology. For example the graphic beta move is a crossing smoothing move in the knot diagrams sector of GLC.

It would be interesting to explore in more detail this distinction – topology does not compute vs topology does compute – for the benefit of all these diagrammatic formalism. The distinction may turn out to be more subtle. For example, Kauffman' bracket polynomial algorithm [9] uses a combination of skein relations and smoothing moves, thus, as remarked in [2], section 5, "This is similar to allowing free beta reduction in the lambda calculus graphs. [...] An analogous situation could occur in GLC where one would need the average over all the results of the many branching calculations." There exist probably overarching formalisms, which blend the two roles of topology in knot diagrammatics related to logic, waiting to be discovered.

References

- [1] M. Buliga and L. H. Kauffman. Chemlambda, universality and self-multiplication. [arXiv:1403.8046](#), to appear in Proceedings of Artificial Life Conference Summer 2014. 2014.
- [2] M. Buliga, L. H. Kauffman, GLC actors, artificial chemical connectomes, topological issues and knots. [arXiv:1312.4333](#)
- [3] M. Buliga, Chemical concrete machine, (2013), [arXiv:1309.6914](#)
- [4] , M. Buliga, Graphic lambda calculus, Complex Systems 22, 4 (2013), 311-360, [arXiv:1305.5786](#)
- [5] A. Y. Carmi, D. Moskovich, Tangle Machines I: Concept, [arXiv:1404.2862](#)
- [6] A. Y. Carmi, D. Moskovich, Tangle Machines II: Invariants, [arXiv:1404.2863](#)
- [7] L. H. Kauffman, Knot Automata. Proceedings of The Twenty-Fourth International Symposium on Multiple-Valued Logic, 1994, Boston, Massachusetts , p. 328 - 333, ([pdf](#))
- [8] L. H. Kauffman, Knot Logic. In "Knots and Applications" ed. by L. Kauffman, World Scientific Pub.(1994), pp. 1-110.([pdf](#))
- [9] L. H. Kauffman, "Knots and Physics", World Scientific Pub. (1991, 1993, 2001, 2013).
- [10] L. H. Kauffman and S. J. Lomonaco. Quantum diagrams and quantum networks, [arXiv:1404.4433](#) to appear in Proceedings of Spie Conference, Baltimore, May 2014. 2014